Attorney's Docket No.: 10559/619001 / P12858

APPLICATION

FOR

UNITED STATES LETTERS PATENT

TITLE: CONGESTION MANAGEMENT FOR HIGH SPEED QUEUING

APPLICANT: MARK B. ROSENBLUTH, DEBRA BERNSTEIN AND GILBERT WOLRICH

CERTIFICATE OF MAILING BY EXPRESS MAIL

Express Mail Label No. EL870691168US

I hereby certify that this correspondence is being deposited with the United States Postal Service as Express Mail Post Office to Addressee with sufficient postage on the date indicated below and is addressed to the Commissioner for Patents, Washington, D.C. 20231.

December 17, 2001
Date of Deposit

Signature

Gabe Lewis
Typed or Printed Name of Person Signing Certificate

CONGESTION MANAGEMENT FOR HIGH SPEED QUEUING

TECHNICAL FIELD

This invention relates to congestion management for high speed queuing.

BACKGROUND

Some network devices such as routers and switches have line speeds that can be faster than 10 Gigabits. For maximum efficiency the network devices should be able to process data packets, including storing them to and retrieving them from memory at a rate at least equal to the line rate. Network devices implement congestion avoidance algorithms such as Weighted Random Early Discard (WRED) to preserve chip resources and to regulate packet flow by probabilistically dropping packets as output queue lengths increase beyond predefined limits. The count of packets or buffers for each queue should be observable for all output queues.

BRIEF DESCRIPTION OF THE DRAWINGS

FIG. 1 is a block diagram of a network system.

FIG. 2 is a block diagram of a network device used in the system of FIG. 1.

FIG. 3 is a block diagram of an output queue.

FIG. 4 is a block diagram of a datapath in a processor.

FIG. 5 is a block diagram of entries in a CAM device to track queue descriptors.

FIG. 6 is a flow diagram of a queue description update process.

DETAILED DESCRIPTION

Referring to FIG. 1, a network system 10 for processing data packets includes a source of data packets 12 coupled to a network device 14 and a destination for data packets 16 coupled to the network device 14. The network device 14 includes a processor 18 and a memory 20 having memory data structures 22 configured to receive, store and forward the data packets to a specified destination. Example network devices 14 are network switches, network routers and other network devices. The source of data packets 12 can include, for example, other network devices (not shown) connected over a communications path (not shown) operating at high data packet transfer line speeds. Examples of such communications paths include as an example, an optical carrier (OC)-192 line or a 10-Gigabit Ethernet line. The destination of data packets 16 may also include other network devices as well as a similar network connection.

Referring to FIG. 2, the network device 14 includes memory 20 coupled to the processor 18. The memory 20 provides output

queues 22 and their corresponding queue descriptors 24 in a queue array 26. The memory 20 includes a queue manager programming engine 27 and Content Addressable Memory (CAM) 28.

Upon receiving a data packet from the source 12 (of FIG. 1), the processor 16 performs enqueue and dequeue operations to process the packet. An enqueue operation adds information that has arrived in a data packet to one of the output queues 22 and updates its corresponding queue descriptor 24. A dequeue operation removes information from one of the output queues 22 and updates the corresponding queue descriptor 24, allowing the network device 14 to transmit the information to the appropriate destination 16.

Enqueue and dequeue operations for a large number of output queues 22 in memory 20 at high bandwidth line rates can be accomplished by storing some of the queue descriptors 24 in a cache 42 at the processor's memory controller 44. Commands to perform enqueue or dequeue operations check whether queue descriptors 24 corresponding to the enqueue or dequeue commands are stored in the cache 42. When an enqueue or a dequeue operation is required with respect to a queue descriptor 24 that is not in the cache 42 (a cache miss), the processor 18 issues commands to the memory controller 44 to move a queue descriptor 24 from the cache 42 to the memory 20 and to fetch a new queue

descriptor 24 from memory 20 for storage in the cache 42. In this manner, modifications to a queue descriptor 24 made by enqueue and dequeue operations occur in the cache 42 and are copied to the corresponding queue descriptor 24 in memory 20 upon removal of that queue descriptor 24 from the cache 42.

A sixteen entry CAM 28 with a Least Recently Used (LRU) replacement policy is used to track sixteen queue descriptors 24 that are cached in a queue array 46 of the memory controller 44.

Using a network device 14 implemented as hardware-based multi-threaded processor having multiple microengines (not shown), each CAM entry stores a 32 bit value. Microengines each maintain a plurality of program counters in hardware and states associated with the program counters. Effectively, a corresponding plurality of sets of threads can be simultaneously active on each of the microengines while only one is actually operating at any one time. During a lookup operation CAM entries are compared against a source operand. All entries are compared in parallel, and the result of the lookup is a 6-bit value. The 6-bit result includes a 2-bit code concatenated with 4-bit entry number. Possible results of the lookup are three fold. A first result is a miss where the lookup value is not in the CAM 28 and the entry number is the Least Recently Used (LRU) entry which can be used as a suggested entry to replace. The

second result can be a hit where the lookup value is in the CAM 28 and state bit is clear, and the entry number is an entry which has matched. In addition, a locked result may occur where the lookup value is in the CAM 28, the state bit is set and the entry number is an entry. The state bit is a bit of data associated with the entry, used typically by software. There is no implication of ownership of the entry by any context.

Referring to FIG. 3, an example of an output queue 22 and its corresponding queue descriptor 24 is shown. The output queue 22 includes a linked list of elements each of which has a pointer 32 to the next element 30 in the output queue 22. Each element in the linked list 30 includes an address 34 of information stored in memory 20 that the linked list element represents. The queue descriptor 24 includes a head pointer 36, a tail pointer 38 and a count 40. The head pointer 36 points to the first linked list element 30 of the queue 22, and the tail pointer 38 points to the last linked list element 30 of the output queue 22. The count 40 identifies a number (N) of linked list elements 30 in the output queue 22.

Referring to FIG. 4, details of an arrangement of the CAM 28 in a datapath 70 of the network device 10 implemented as a processor are shown. A General Purpose Register (GPR) file 72 stores data for processing elements 74. The CAM receives

operands as any other processing element 74 would. Operational code (Opcode) bits in an instruction select which processing element 74 is to perform the operation specified by the instruction. In addition, each of the processing elements 74, including the CAM 28, can return a result value from the operation specified by the instruction back to the GPR file 72.

Referring to FIG. 5, a CAM 28 includes an array 76 of tags having a width the same as the width of the GPR file 72. Associated with each of the tags in the array are state bits 78. During a CAM lookup operation, a value presented from the GPR file 72 is compared, in parallel, to each of the tags in the array 76 with a resulting match signal 80 per tag. The values in each tag were previously loaded by a CAM load operation. During the CAM load operation the values from the GPR file 72 specify which of the tags in the array 76 to load and a value to load. Also during the CAM load operation the state information to load is part of the operand.

The result of the CAM lookup is written to a destination GPR file 82 and includes three fields. A hit/miss indication field 84, an entry number field 86 and a state information field 88. If a "hit" occurs, the entry number field 86 is matched. In a "miss," the entry number field 86 is the Least-Recently-Used (LRU) entry.

The following instructions are one example of instructions used to manage and use the CAM 28:

Load (Entry_Number, Tag_Value, State Value)
Lookup (Lookup_Value, Destination)
Set_State (Entry_Number, State_Value)
Read_Tag (Entry_Number, Destination)
Read_State (Entry_Number, Destination)

The LRU Logic 90 maintains a time-ordered list of the CAM 28 entry usage. When an entry is loaded or matches on a lookup, it is marked as MRU (Most Recently Used). A lookup that misses does not modify the LRU list.

If a queue descriptor 24 required for either an enqueue or dequeue is not in the queue array 46, the queue manager programming engine 26 issues a write-back to memory of the LRU entry, followed by a fetch to the same entry, before issuing the enqueue or dequeue command. If the CAM 28 lookup indicates that the needed queue descriptor 24 is already in the queue array 46, then the enqueue or dequeue command is issued without replacing an entry.

Each enqueue command increments the count 40 of packets or buffers for a particular output queue 22. A dequeue command decrements the count 40 of packets or buffers when a pointer to the buffer descriptor 24 at the head of the output queue 22 is updated.

The microengine (in a processor containing multiple microengines) tasked with congestion avoidance reads the queue descriptors 24 from memory 20 to determine the length (count word 40) of each output queue 22. The queue descriptors 24 for highly used output queues 22 can remain in the queue array 46 of the memory controller 44 for an infinitely long time period. A Write_Q_Descriptor_Count Command is issued by the queue manager programming engine 26 after the enqueue or dequeue command, when the entry used "hits" the CAM 28. the format of the command is:

Write_Q_Descriptor_Count (address, entry).

The command uses two parameters, i.e., address and entry, and keeps the countfield 40 for all queue descriptors 24 current in memory 20 for the microengine implementing congestion avoidance. The write of a single word containing the queue count information for entries that hit in the query array 46 in the cache 42 replaces a write-back of two or three words when a new entry needs to be fetched.

Referring to FIG. 6, a write queue descriptor process 100 includes receiving (102) an address and a queue subsequent to an enqueue or dequeue command. The process 100 maintains (104) a count field for all queue descriptors current in memory for the microengine implementing congestion avoidance. The process 100 writes (106) a single word containing the queue count

information for the queue entry that hits the queue array in the cache.

It is to be understood that while the invention has been described in conjunction with the detailed description thereof, the foregoing description is intended to illustrate and not limit the scope of the invention, which is defined by the scope of the appended claims. Other aspects, advantages, and modifications are within the scope of the following claims.